The Visual Computer

Right-triangular subdivision for texture mapping ray-traced objects

Uğur Akdemir, Bülent Özgüç, Uğur Güdükbay, Alper Selçuk

Department of Computer Engineering and Information Science, Bilkent University, Bilkent, 06533 Ankara, Turkey
e-mail: {akdemir, ozguc}@bilkent.edu.tr
gudukbay@cs.bilkent.edu.tr
alpers@microsoft.com

The introduction of global illumination and texture mapping enabled the generation of high-quality, realistic looking images of computer graphics models. We describe a fast and efficient 2D texture mapping algorithm that uses triangle-to-triangle mapping, taking advantage of mapping an arbitrary triangle to a right triangle. This requires fewer floating point operations for finding the 2D texture coordinates and little preprocessing and storage. Texture mapping is combined with ray tracing for better visual effects. A filtering technique alternative to area sampling is developed to avoid aliasing artifacts. This technique uses linear eye rays, and only one eye ray per pixel is fired. A uniform supersampling filtering technique eliminates aliasing artifacts at the object edges.

Key words. Texture mapping – Ray tracing – Area sampling – Filtering – Summed area tables – Sweep surfaces

Correspondence to: U. Güdükbay

1 Introduction

Modeling every minute detail of an object to give realistic effects is a very difficult and inefficient process. We overcome this difficulty with texture mapping, which can increase surface detail at a relatively low cost. Ray tracing is one of the most powerful methods in computer graphics for generating realistic images. In this study, we combine our texture mapping method with area sampling ray tracing for reducing aliasing.

The objects are produced by a tool called T♭ ('Topology♭ook'), which was developed at Bilkent University [1]. The objects are created by sweeping a 2D contour curve that can vary around a 3D trajectory curve. Contour and trajectory curves can be Bézier or free curves. A lot of interesting objects can be created with T♭.

We have implemented both 2D and 3D procedural texture mapping, which is easily applicable to any kind of surface. Procedural texturing functions do not depend on the surface geometry or the surface coordinate system. Therefore, procedural mapping gives good results for complex general sweep objects. We have embedded a few procedural mapping routines in our system to increase the realism of the scenes combined with 2D texture mapping. However, we do not explain our procedural texture-mapping implementation since it uses the well-known *noise* function [13].

In 2D texture mapping, a texture or an image is fitted onto a 3D surface. One popular method maps triangles from the texture to the triangles on the model. We have developed a simple and efficient method for triangle-to-triangle mapping. Since the mapping takes advantage of mapping an arbitrary triangle to a right triangle, it requires many fewer floating point operations to find the 2D texture coordinates, and it does not incur the preprocessing and storage costs that other texture mapping algorithms do.

Two-dimensional texture mapping causes severe aliasing unless it is filtered. Unfortunately, classical naive ray tracing is a point-sampling technique that creates difficulties for filtering. We suggest a filtering technique for ray tracing as an alternative to area sampling to reduce aliasing; it is simpler than other area sampling methods. In our implementation, we use a prefiltering technique, viz. repeated integration filtering [10], which is a generalization of summed area tables [5].

This paper assumes that the reader is familiar with ray tracing, texture mapping, and antialiasing. See [7] for an excellent reference about ray tracing. A

The Visual Computer (1998) 14:445–454

survey of texture mapping and filtering techniques to eliminate aliasing is presented in [11].
The rest of the paper is organized as follows. In Sect. 2, our 2D texture mapping method is described. Then in Sect. 3, our alternative filtering technique for ray tracing is explained. In Sect. 4, the results of our implementation is given, and in Sect. 5 the conclusions are discussed.

2 The 2D texture mapping process

The triangle is one of the most used geometric primitives in computer graphics for representing complex surfaces. Most graphics engines are optimized for rendering triangles. Therefore, triangulating graphics models before rendering is one of the most popular optimizations.
Since the 2D texture mapping algorithm presented in this paper is based on triangle-to-triangle mapping, it accepts 3D triangulated objects as input. If the 2D parametric coordinates of the vertices of the triangles are known in advance, the texture mapping algorithm produces better results.

2.1 Previous work

2.1.1 Barycentric coordinates

Barycentric division is used to subdivide a line segment into equal lengths. To form barycentric coordinates in a triangle, each edge of a triangle is subdivided into equal lengths and axes that are parallel to the edges. Barycentric coordinates in two-dimensions can be used for mapping an arbitrary triangle to another arbitrary triangle [4]. A point is represented as (u, v) in Cartesian coordinates and as (L_1, L_2, L_3) in barycentric coordinates. The mapping from barycentric coordinates to Cartesian coordinates is as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix}. \tag{1}$$

This mapping can be inverted as:

$$\begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{2}$$

This method involves calculating the barycentric coordinates in two dimensions for each point that is to be mapped. Substantial amounts of preprocessing and storage are needed for each triangle.

2.1.2 Extension from convex quadrilateral inverse mapping to triangle inverse mapping

A convex quadrilateral inverse mapping algorithm is given in [8]. The parametric values (u, v) are calculated by the algorithm for obtaining the location of a point within a convex quadrilateral. This coordinate pair represents the location of the point with respect to the four edges, taken as pairs of coordinate axes ranging from 0 to 1. To achieve the triangle-to-triangle inverse mapping, the triangle is passed to the algorithm by doubling the last vertex in order to give the routine four points to work with. At the doubled vertex, all values of one parameter converge. Hence, the precision of the mapping process decreases.
The derivation for plane-dependent constants is rather complex, and it needs a substantial amount of preprocessing and storage. Seventeen floating-point numbers have to be stored for each triangle. According to the calculations in [8], 34 multiplications and two square root operations are needed for each (u, v) pair.

2.1.3 Parameterization of triangles

Heckbert [11] gives a 2D parameterization for triangles defined in three dimensions. The equation for parameterization is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{3}$$

The nine variables, A through I, are calculated from the Cartesian coordinates of the inversely mapped triangle by solving three sets of equations with three unknowns by using the 3D coordinates of the vertices of the triangle. In order to directly parameterize a point on a triangle, Eq. 3 can be inverted as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{4}$$

This is one of the most commonly used methods for triangle-to-triangle mapping. Like the other methods mentioned, this method has a high preprocessing cost and it needs a substantial amount of storage space for the 3×3 matrices of each triangle. Assuming 8 bytes for each floating-point number in the 3×3 matrix, we need 720000 bytes of memory for 10000 triangles.

2.2 Our method

All of the methods mentioned need substantial amounts of preprocessing and storage space.

If we assume that the 2D texture image is rectangular, which is almost always the case, then we can subdivide this image into triangles by using only right triangles (Fig. 1). The rectangular texture image is subdivided into similar rectangles along the u–v directions. Then these rectangles are subdivided into two right triangles. By taking advantage of mapping an arbitrary triangle to a right triangle, we propose a fine, fast, and efficient triangle-to-triangle mapping. The proposed 2D-texture mapping method works for any object represented by triangles. If the 2D parametric coordinates of the objects are known in advance, the mapping will give better visual results.

The first step of the method is to subdivide the texture image into right triangles in order to find the corresponding triangles on the surface area as seen in Fig. 1.

The next step is to find the corresponding point H_2 on T_2 for a point H_1 on T_1 (Fig. 2).

To make a fine interpolation, we preserve the following ratios for both of the triangles seen in Fig. 2.

$$r_{BA} = \frac{|B_1 I_{B_1 A_1}|}{|B_1 A_1|} = \frac{|B_2 J_{B_2 A_2}|}{|B_2 A_2|} \quad \text{and}$$

$$r_{BC} = \frac{|B_1 I_{B_1 C_1}|}{|B_1 C_1|} = \frac{|B_2 J_{B_2 C_2}|}{|B_2 C_2|}. \tag{5}$$

Since the medians of a triangle meet at its center of gravity, the centers of gravity of the triangles are preserved when the ratios in Eq. 5 are preserved. By preservation of the center of gravity, we mean the center of gravity in a right triangular texture patch maps to the center of gravity in the general triangle that is texture mapped. Although the distortion characteristics of a texture mapping algorithm depends largely on the kinds of objects that are texture mapped, not preserving the center of gravity increases the distortion. This can be described in terms of *homogeneity of the resolution* and *aspect ratio*, which are the two criteria used to evaluate a mapping algorithm [3]. Homogeneity of resolution is determined as the ratio of the maximum and minimum values of the arc lengths measured on the mapping along a principal axis as a result of the corresponding motions on the texture patch. It gives a quantization of the effective scaling of the mapping along principal axis directions, which ideally should be constant. The aspect ratio is the maximum value of the proportion of the arc lengths along the principal axes corresponding to perpendicular motions along the principal axes on the texture patch, which are measured at various points on the mapping.

This can be explained better with an example. Assume that we are trying to map a triangular texture patch to a triangle of the same shape and size. If the center of gravity is preserved, then the homogeneity of the resolution and the aspect ratio of the mapping will not be disturbed. This means that these criteria will be optimal for this example. If the center of gravity is not preserved, these criteria will not be optimal. The derivations in the previous mapping methods are either too costly to preserve the centers of gravity or they do not preserve the centers of gravity.

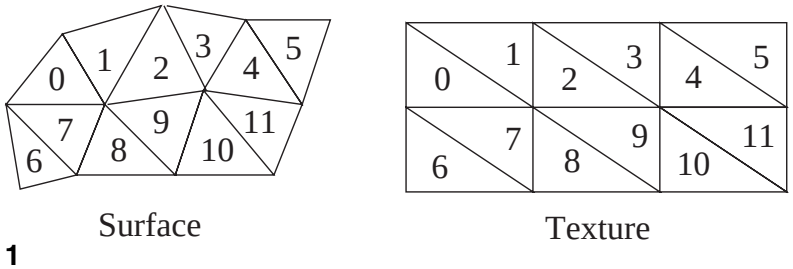
Point H_1 is represented by a 3D coordinate system since it is a point on a 3D object. The following equation can be written in terms of x, y, z coordinates to find r_{BC} in Eq. 5.

$$\begin{bmatrix} x_{I_{B_1 C_1}} \\ y_{I_{B_1 C_1}} \\ z_{I_{B_1 C_1}} \end{bmatrix} = (1 - r_{BC}) \cdot \begin{bmatrix} x_{B_1} - x_{C_1} \\ y_{B_1} - y_{C_1} \\ z_{B_1} - z_{C_1} \end{bmatrix} + \begin{bmatrix} x_{C_1} \\ y_{C_1} \\ z_{C_1} \end{bmatrix}. \tag{6}$$

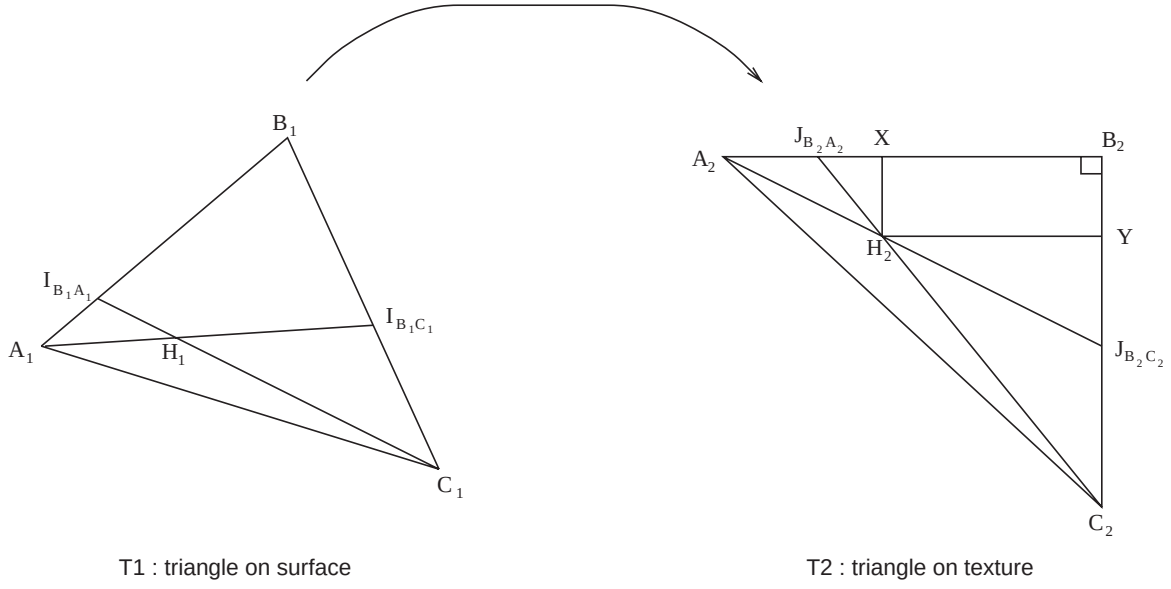
$\overrightarrow{A_1 H_1}$ and $\overrightarrow{H_1 I_{B_1 C_1}}$ have the same direction. If T_1 is on neither the x nor the y-plane, then the following equation can be written.

$$\frac{x_{H_1} - x_{A_1}}{y_{H_1} - y_{A_1}} = \frac{(1 - r_{BC}) \cdot (x_{B_1} - x_{C_1}) + x_{C_1} - x_{H_1}}{(1 - r_{BC}) \cdot (y_{B_1} - y_{C_1}) + y_{C_1} - y_{H_1}}. \tag{7}$$

If H_1 and A_1 coincide, H_1 is mapped directly to A_2. If T_1 is on the x-plane, the xs should be replaced with zs, and if T_1 is on the y-plane, the ys should



1



2

Fig. 1. 2D texture mapping

Fig. 2. Mapping from an arbitrary triangle to a right triangle

be replaced with zs in this equation. After arranging the terms, we get:

$$r_{BC} = 1 - \frac{(x_{H_1} - x_{A_1}) \cdot (y_{C_1} - y_{H_1}) - (y_{H_1} - y_{A_1}) \cdot (x_{C_1} - x_{H_1})}{(y_{H_1} - y_{A_1}) \cdot (x_{B_1} - x_{C_1}) - (x_{H_1} - x_{A_1}) \cdot (y_{B_1} - y_{C_1})}. \quad (8)$$

The value of r_{BA} can be found similarly. If H_1 is on $|A_1C_1|$, i.e., both r_{BC} and r_{BA} are 1, then the corresponding point H_2 on the hypotenuse of T_2 can be found by calculating r_{AC} similarly.

After calculating the ratios in Eq. 5 on T_1, all we need to do is to find H_2 on T_2 by preserving these ratios. Since the coordinates of the vertices of T_2 are known beforehand, calculating $|B_2Y|$ and $|B_2X|$ is enough to find the 2D coordinates of point H_2. $|B_2Y|$ and $|B_2X|$ can be found by using the similarity relation $J_{B_2A_2}\overset{\Delta}{X}H_2 \sim J_{B_2A_2}\overset{\Delta}{B_2}C_2$ between the triangles. After arranging the terms and some simplifications to reduce the number of multiplication operations, we get:

$$|B_2Y| = \frac{|B_2C_2| \cdot (r_{BC} \cdot r_{BA} - r_{BC})}{r_{BC} \cdot r_{BA} - 1}. \quad (9)$$

If H_1 is on $|A_1C_1|$, then:

$$|B_2Y| = r_{AC} \cdot |B_2C_2|, \tag{10}$$

$$|B_2X| = (1-r_{AC}) \cdot |B_2A_2|. \tag{11}$$

Because of the floating-point intersection calculations, the probability that both r_{BC} and r_{BA} will be 1, i.e., H_1 will be exactly on $|A_1C_1|$, is very low. Therefore, if we ignore the extra calculations for hit points on $|A_1C_1|$ and a few comparison operations, we need 16 floating-point multiplications/divisions to find the real 2D coordinates in the texture space. One other nice advantage of this method is that almost no preprocessing and storage are needed. Only three floating-point numbers, the edge lengths of T_2, need to be stored for each texture image.

3 Anti-aliasing for ray tracing

Although ray tracing is one of the best methods for global illumination, it causes severe aliasing when used with texture mapping because it is a point-sampling method. Therefore, we must either sample areas to use texture mapping with ray tracing or use a filtering technique such as an alternative area sampling to eliminate aliasing. Since area sampling is too expensive, we choose to implement the alternative filtering technique described in the sequel.

3.1 Methods for area sampling

Ray tracing with beam [9], cone [2], and pencil [14] rays are methods for area sampling, but they require complex intersection and clipping calculations. Whitted [16] suggests using a pyramid instead of a linear eye ray that is defined by the four corners of a pixel. These pyramids are intersected with the objects. Therefore, this method is computationally very expensive. Sung [15] suggests the area sampling buffer (ASB), which allows the use of z-buffer hardware for area sampling in a ray-tracing style. This method is superior to the other methods mentioned and it takes advantage of specialized hardware, but it suffers from the limitations of the z-buffer for transparent objects.

3.2 Our implementation

Except the ASB method, all of the area sampling methods mentioned replace linear rays with a geometric object. The intersections are calculated with respect to these geometries. Complex calculations are needed for clipping and intersections.
In our implementation, only one linear eye ray per pixel is used. A ray that is shot from the view point is traced in the environment. Therefore, all of the intersection calculations remain the same as in classical ray tracing.
To achieve the filtering by using a classical ray tracer, we used three buffers, each holding the necessary hit information for a screen row. When three row buffers have been filled with the hit information obtained from the ray tracer, we begin calculating the color value for the middle row. The color value for a pixel is calculated by beginning from the node at the maximum reflection depth level to the primary ray level. The color values are transmitted to a lower reflection level.
The linked list representation for a typical row buffer for three pixels is shown in Fig. 3. Refraction levels grow vertically and reflection levels grow horizontally.
Our implementation of the filtering technique for ray tracing consists of three parts:

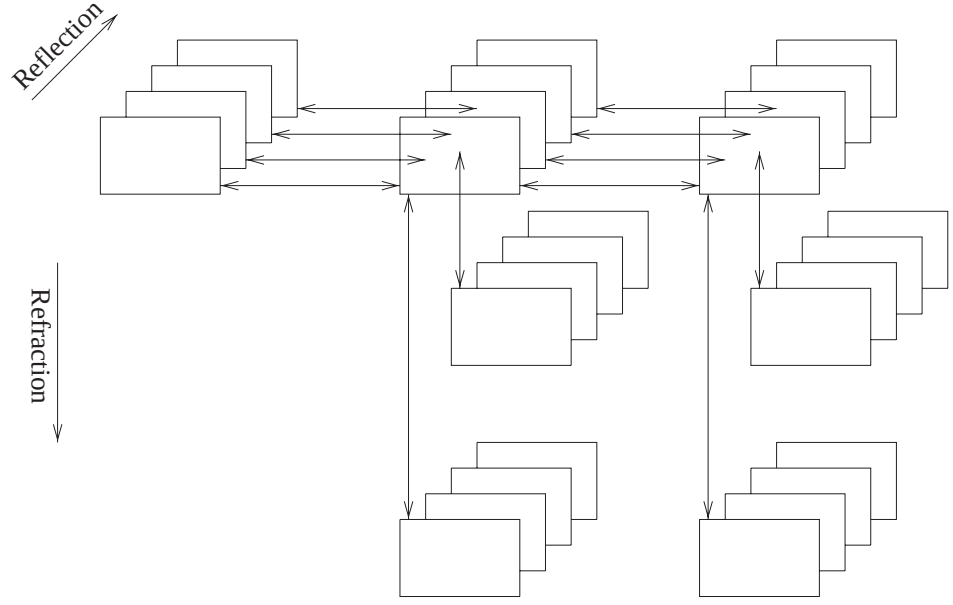
1. Construction of intersection trees
2. Shading
3. Shifting rows of intersection trees.

3.2.1 Construction of intersection trees

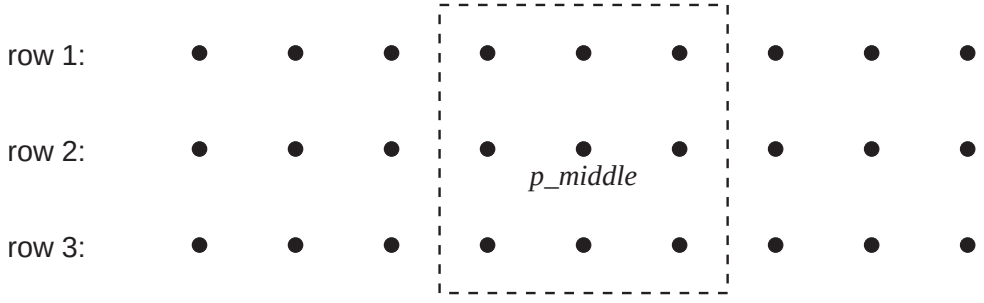
An intersection tree is constructed for a row as seen in Fig. 3. Each node of an intersection tree contains the minimum necessary information, i.e., object id, coordinates of the intersection point, and texture map parameters if they exist.
At least three rows of intersection trees must be created and stored for the filtering process.

3.2.2 Shading

Shading is done after three row buffers have been filled with intersection trees. In Fig. 4, the color value for the filtering operation, i.e., the diffuse color, for node *p_middle* is calculated by filtering the area surrounded by the box, if the nodes are on the same reflection and refraction depths and



3



4

Fig. 3. A linked list representation of a row buffer

Fig. 4. Our filtering process for ray tracing

they hit on the same area that will be textured. If one of these conditions does not hold, we get the diffuse color value of the object at that point without filtering. To this diffuse color value, we add the transmitted color value coming from a higher reflection level and a refraction level if it exists. If the node is at the maximum depth level, the transmitted color value is zero. If the reflected ray goes out of the space, then the transmitted color value for reflection is the background color. Local color is determined by the illumination of the light sources. If the object for a node is a refracting surface, we do the same operations for the refraction list and contribute the color value from the refraction to that point. We transmit the local color value to a lower reflection level. This operation is repeated until the actual pixel color is found.

Although our filtering technique eliminates the aliasing artifacts on the 2D texture maps on the objects, aliasing artifacts still occur at the edges of the objects. To alleviate these aliasing artifacts, we use *uniform supersampling*. A 3 n×3 n image can be dwarfed into an n×n image easily, if we assume that nine rays per pixels are used. Three rows

in the high-resolution image are used to determine a row in the low-resolution image. The color values of nine pixels in the high-resolution image are weighted and averaged to determine the color value of a pixel in the low-resolution image. The weights are assigned according to the area that a pixel in a high-resolution image contributes to a pixel in a low-resolution image, i.e. 1/4 for the pixel in the middle, 1/16 for each of the four pixels at the corners, and 1/8 for each of the remaining four pixels. Although this method requires more processing time than the original method, higher-quality images without staircases appearing at the edges are produced.
As an alternative to uniform supersampling, an adaptive supersampling, which is less expensive, could be used.

3.2.3 Shifting rows of intersection trees

After we are finished with row 2, we shift row 2 to the place of row 1 and row 3 to the place of row 2. Then row 3 is filled as previously described.

3.3 Acceleration

Most of the time in a ray tracer is spent on intersection calculations. To reduce these calculations, we put the sweep objects into bounding boxes. Simple bounding box intersection tests precede the costly intersection tests for the objects inside these bounding boxes.
We need to represent the sweep objects with a large number of triangles to get well-formed surfaces. We need to take advantage of the spatial coherency of this triangle-based representation to speed up the intersection calculations. We implemented the spatially enumerated auxiliary data structure (SEADS) [6] to take advantage of the spatial coherency. SEADS involves dividing the bounding boxes into 3D grids (voxels). The width, length, and height of each grid are determined by the shape and orientation of the object. Since the objects are created by sweeping, they have a regular mesh structure, and this fact is utilized in implementing SEADS. The grids are divided with respect to the number of contour and trajectory curves of the sweep object. In this way, each grid contains a maximum of about six triangles.
To take the advantage of this data structure, the voxels along a ray path should be traversed efficiently. These voxels are traversed with the 3D digital differential analyzer (3DDDA) [6]. The 3DDDA algorithm uses only integer operations. If there is more than one object, their bounding boxes can be placed in a global grid to make the ray-box intersection tests more efficient. A moderate sweep object consists of about 1000 triangles. Instead of checking ray triangle intersections with all these triangles, we only do a box intersection test and a maximum of about six ray triangle intersections. This is a very considerable improvement. Besides the 3DDDA method, any accelerating method developed for a classical ray tracer that takes advantage of spatial coherency can be embedded in our implementation.

4 Results

Some 512×512 resolution examples from our implementation are shown in Figs. 5–7. Figure 5 was produced by point sampling. Figure 6 was produced by filtering the textures. Note the aliasing of the textures both on the textures and on their reflections in Fig. 5.
All of the objects in Figs. 5–7 were created by TŁ [1]. The vase and the cup were created by rotating a Bézier curve around a rotational axis. The handle part of the cup was created by sweeping a circle-shaped contour curve around a 'U'-shaped trajectory curve. This handle was then combined with the cup. The vase consists of 7200 triangles. The cup consists of 8400 triangles. All of the objects seen in Figs. 5–7 are reflective and opaque. The left wall is a mirror, and the floor has a moderate reflectivity. The vase and cup have low reflectivities. The floor is also 2D texture mapped. It took about 15 min to create Fig. 5 and 20 min to create Fig. 6 on a networked SUN Sparc ELC workstation. Figures 8 and 9 give two more images produced with our implementation.
Our texture mapping algorithm has a number of advantages over other triangle-to-triangle texture mapping algorithms. It calculates the 2D texture coordinates of the points to be mapped using a small number of floating-point operations. It does not incur any preprocessing or storage costs since the texture coordinates are calculated on-the-fly from the ratios between the edges of the triangles.

5

7

6

Fig. 5. A point sampled image

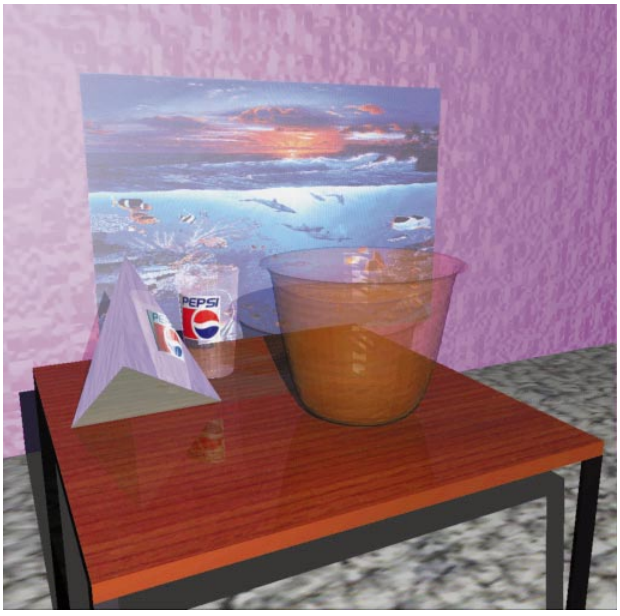
Fig. 6. A filtered image

Fig. 7. A filtered and supersampled image

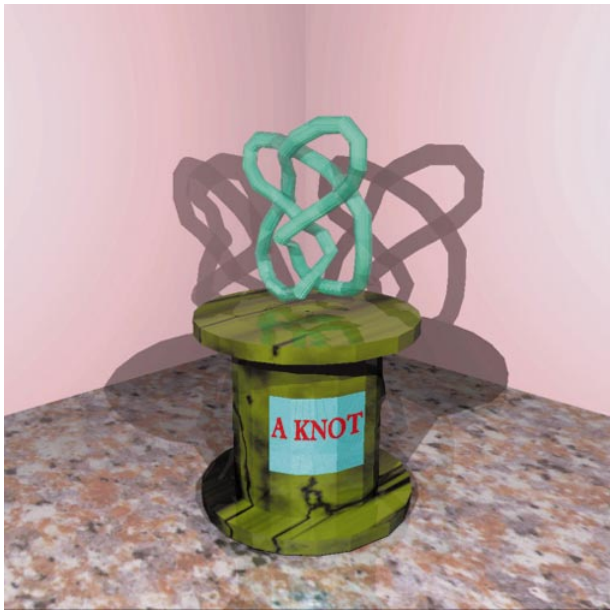
Besides, since it is very easy to preserve the centers of gravity of the triangles with our method, some unwanted visual effects, introduced when the centers of gravity are not preserved, disappear. This is a very costly operation for the other algorithms.

Since we area sample only the 2D texture maps on the objects, aliasing effects still occur at the edges of the objects. To alleviate these aliasing effects, we used supersampling. A high-resolution version of Fig. 6 is produced, and then it is supersampled. The result of supersampling is shown in Fig. 7. Note how the staircases seen at the edges disappear.

We observed that, as the sizes of the textures become larger, the summed area tables become big-



8



9

Fig. 8. A filtered and supersampled image including semitransparent objects

Fig. 9. A semitransparent, depth-modulated, sweep object on top of a knot on a marble column (filtered)

ger and the program needs more space. Therefore, using big textures slows down the execution.

5 Conclusion

We have introduced our method for combining texture mapping with ray tracing in order to generate highly realistic scenes. We developed our scenes by using sweep surfaces, and the methods we proposed can be applied to any objects represented by triangles.

We proposed a fine, fast, and efficient texture mapping method based on triangle-to-triangle mapping. Since the method utilizes the advantage of mapping from an arbitrary triangle to a right triangle, it requires many fewer floating-point operations, and it does not require the preprocessing and storage that other texture mapping methods do.

Area sampling in ray tracing is essential for filtering textures. We also proposed a filtering technique as an alternative to area sampling in ray tracing. The method we proposed delays shading until enough row buffers are filled with necessary hit information. The advantage of this implementation is that the intersection calculations use linear eye rays, and only one linear eye ray per pixel. This method can also be improved for producing smooth shadow edges. We took advantage of the spatial coherency of the triangles forming the objects in a scene by 3D grid partitioning. To eliminate the aliasing artifacts at the edges of the objects, we used a uniform supersampling filtering technique.

References

1. Akman V, Arslan A (1992) Sweeping with all graphical ingredients in a topological picturebook. Comput & Graph 16: 273–281
2. Amanatides J (1984) Ray tracing with cones. Comput Graph (Proceedings of SIGGRAPH '84) 18:129–135
3. Bier EA, Sloan KR (1986) Two-part texture mapping. IEEE Comput Graph Appl 6:40–53
4. Celniker G, Gossand D (1991) Deformable curve and surface finite-elements for free-form shape design (1991) Comput Graph (Proceedings of SIGGRAPH '91) 25:257–266
5. Crow FC (1984) Summed area tables for texture mapping. Comput Graph (Proceedings of SIGGRAPH '84) 18:207-212
6. Fujimoto A, Tanaka T, Iwata K (1986) ARTS: Accelerated ray tracing system. IEEE Comput Graph and Appl 6:16–26

7. Glassner AS (1989) An introduction to ray tracing. Academic Press, San Diego, Calif
8. Haines E (1991) Essential ray tracing algorithms (1991) In: Glassner A (ed) An introduction to ray tracing, Academic Press San Diego, Calif pp 33–77
9. Heckbert PS (1986) Filtering by repeated integration. Comput Graph (Proceedings of SIGGRAPH '86) 20:315–321
10. Heckbert PS (1986) Survey of texture mapping. IEEE Comput Graph and Appl 6:56–67
11. Heckbert PS, Hanrahan P (1984) Beam tracing polygonal objects. Comput Graph (Proceedings of SIGGRAPH '84) 18:119–128
12. Peachey DR (1985) Solid texturing of complex surfaces. (1985) Comput Graph (Proceedings of SIGGRAPH '85) 19:279–286
13. Perlin K (1985) An image synthesizer (1985) Comput Graph (Proceedings of SIGGRAPH '85) 19:287–296
14. Shinya M, Tokiichiro T (1987) Principles and applications of pencil tracing. Comput Graph (Proceedings of SIGGRAPH '87) 21:45–54
15. Sung K (1992) Area sampling buffer: tracing rays with *Z*-buffer hardware. Proceedings of Eurographics '92 11:299–310
16. Whitted T (1980) An improved illumination model for shaded display. Commun ACM 23:343–349

UĞUR AKDEMIR was born in Ankara, Turkey, in 1969. He received his B.Sc. and M.Sc. from Bilkent University in 1991 and 1993 respectively, both in Computer Engineering and Information Science. Currently, he is working at Tom Sawyer Software, USA. His research interests include global illumination and rendering.

BÜLENT ÖZGÜÇ joined the Bilkent University Faculty of Engineering, Turkey, in 1986. He is a professor of computer science and the dean of the Faculty of Art, Design and Architecture. He formerly taught at the University of Pennsylvania, USA, Philadelphia College of Arts, USA, and the Middle East Technical University, Turkey, and worked as a member of the research staff at the Schlumberger Palo Alto Research Center, USA. For the last 15 years, he has been active in the field of computer graphics and animation. He received his B.Arch. and M.Arch. in architecture from the Middle East Technical University in 1972 and 1973. He received his M.S. in architectural technology from Columbia University, USA, and his Ph.D. in a joint program of architecture and computer graphics from the University of Pennsylvania in 1974 and 1978, respectively. He is a member of ACM Siggraph, IEEE Computer Society, and UIA.

UĞUR GÜDÜKBAY was born in Niğde, Turkey, in 1965. He received his B.Sc. in Computer Engineering from Middle East Technical University in 1987, and his M.Sc. and Ph.D., both in Computer Engineering and Information Science, from Bilkent University in 1989 and 1994, respectively. Then, he pursued a postdoctoral study at University of Pennsylvania, Human Modeling and Simulation Laboratory. Currently, he is an assistant professor at Bilkent University, Computer Engineering and Information Science Department. His research interests include physically based modeling and animation, human modeling, multiresolution modeling, and rendering. He is a member of ACM SIGGRAPH, and IEEE Computer Society.

ALPER SELÇUK was born in Kirşehir, Turkey, in 1973. He received his B.Sc. in Computer Engineering from Hacettepe University in 1995. He received his M.Sc. in Computer Engineering and Information Science from Bilkent University in 1997. Currently he is working at Microsoft, USA. His research interests include multiresolution modeling and rendering.